Naval Research Laboratory

Washington, DC 20375-5000

NRL Memorandum Report 6713

A Suite of Weapon Assignment Algorithms for a SDI Mid-Course Battle Manager

WILLIAM A. METLER AND FRED L. PRESTON

AT&T Bell Laboratories

JIM HOFMANN

Integrated Warfare Technology Branch
Information Technology Division

September 19, 1990

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1990 September 19	

4. TITLE AND SUBTITLE
A Suite of Weapon Assignment Algorithms for a SDI Mid-Course Battle Manager

5. FUNDING NUMBERS
PE - 63223E EC
WU - DN155-097
JO - 55-2354-E0

6. AUTHOR(S)
William A. Metler,* F. L. Preston,*
and James B. Hofmann

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Research Laboratory
4555 Overlook, Ave.,
Washington, DC 20375-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NRL Memorandum
Report 6713

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
SDIO - Office of Chief of Naval Research
Crystal Plaza #5
Arlington, VA 22202

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
*AT&T Bell Laboratories

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

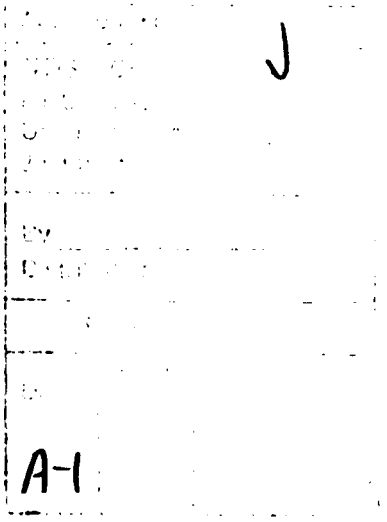
13. ABSTRACT (Maximum 200 words)

In this report, we describe a sequence of engagement problems faced by a mid-course battle manager acting within a three tier defensive system against a ballistic missile threat. Then we discuss a set of weapons allocation algorithms available to solve these problems. Since the weapons allocation problem has been shown to be NP-Complete, the algorithm suite employs heuristics drawn from mathematics, operations research, and artificial intelligence. We present a two stage solution of selection of assets to be defended and of weapons allocation for destroying threat objects targeted at those defended assets. On a different dimension, we also present a two stage solution of aggregated weapons and threat objects (swarms) and the associated single weapon, single object decomposed problems. Generalized and specific objective functions are given which are dependent on the situation and dictate the mix of algorithms available for selection. Control of the suite is based upon a decision framework induced by inputs from a Situation Assessment and Strategic Planning module (SASP).

14. SUBJECT TERMS		15. NUMBER OF PAGES
Weapons allocation	SDI	27
Situation assessment and Strategic planning		16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

CONTENTS



A SUITE OF WEAPON ASSIGNMENT ALGORITHMS FOR A SDI MID COURSE BATTLE MANAGER

1. INTRODUCTION

This document presents a description of the Weapon Allocation and Resource Management (WARM) algorithms developed by AT&T Bell Laboratories and delivered to the Naval Research Laboratory. This suite of heuristics runs in AT&T's Local Experimental Test System (LETS), to develop, study, test, and convincingly demonstrate the operation and effectiveness of the weapon allocation algorithms. This document describes the approach to building the algorithm suite and in detail the mathematics of several candidate algorithms for the suite. Although at present the algorithm suite consists of many "loosely federated" modules of candidate algorithms with shared global data structures, we expect, after analysis in LETS and thorough testing at NRL by NRL personnel, to determine which subset of algorithms will "play" best together and define the conditions for their operation. The objective at the end of the analysis is to bind selected algorithms tightly into a suite for delivery and insertion into the National Test Facility (NTF) testbed.

This introduction places the work on algorithms presented in this document in perspective with previous defensive weapon allocation problems. We first present our notation and assumptions and a decomposition of our problem into successively simpler formulations. We then motivate our approach to the solution of these formulations with a brief survey of classical optimization techniques. Next the algorithm suite itself is presented with detailed descriptions of the specific algorithms we have chosen for development and testing in LETS. Finally we discuss how we will measure the comparative effectiveness of different combinations of algorithms in varying threat scenarios.

The "Prim-Read" deployments of interceptor missiles and their associated firing plans [1] of the late 1950s defend a collection of separated point targets against an attack by an unknown number of sequentially arriving ballistic missiles. The scheme involves deploying and firing the interceptors in a manner that averages the probabilities that each of a prescribed number of attacking weapons destroys the target.

More recently, Burr, et. al. [2], showed that the "Greedy Algorithm" when applied to this Prim-Read problem produces a *globally optimal integral solution*. The objective function for their study is the minimization of the total number of interceptors required to defend K targets against an attack of N missiles, subject to a given upper bound on the maximum target value destroyed per attacking weapon, where neither N nor upper or lower bounds on N are known to the defender in advance.

There are two types of defenses, point and area. Point defenses consist of weapons dedicated to the defense of particular targets, while area defenses consist of weapons used to shoot at objects in an area (or volume) regardless of the object's intended target. The Prim-Read defense focuses on the deployment of a terminal defense and assumes that the firing plans of these defending interceptors will be used in a point defense mode. The Strategic Defense Initiative (SDI) raises the possibility of a three tier defense, i.e., boost, midcourse, and terminal. During boost and midcourse phases, space based weapons potentially may defend against any missiles in range, i.e., an area defense mode. The area defense mode of operation is also characteristic of submarine based interceptors, air defenses involving ground to air and ship to air projectiles, and theater level defenses against short range missiles carrying conventional warheads.

Manuscript approved June 13, 1990.

We study the problem of defensive weapon allocation for area defense *after* the threat is known, whereas the Prim-Read problem studies the problem of defensive weapon deployment for point defense *before* the threat is known. Our problem is to 1) maximize the expected surviving defensive values given M interceptors by determining the best firing schedule over time to defend K target sites against N offensive missiles where 2) any interceptor can kill any missile in range with kill probability dependent on the missile, the interceptor, and the firing time.

We propose a general weapons assignment model, and a sequence of simplifying assumptions to make the problem more tractable. For each special case there is an objective function and a set of constraints. The algorithms that are used to solve each case take advantage of the special circumstances of the situation.

2. *ASSUMPTIONS AND NOTATION*

The general problem is that of a defensive battle manager that must allocate defensive weapons to offensive threats to maximize the surviving value of targeted assets. For purposes of generality we define a "weapon" as having an inventory of "shots" and a "threat object" (TO) to be anything potentially causing damage to defensive assets. The purpose of this section is to introduce overall assumptions and notation.

a. We assume that there are K targets consisting of blue assets (population centers, command and control locations, communications nodes, ICBM launch sites, air fields, etc.), and that each target k has a value, w_k, assigned by the defense. Each blue asset may be targeted by the offense with one of more threat objects (see Figure 1). The assignment (grouping) of threat objects to blue assets is the function of a Situation Assessment and Strategic Planning (SASP) battle management module and this grouping is one input to WARM. The extent of damage caused by a leaking threat object will be discussed in later paragraphs.

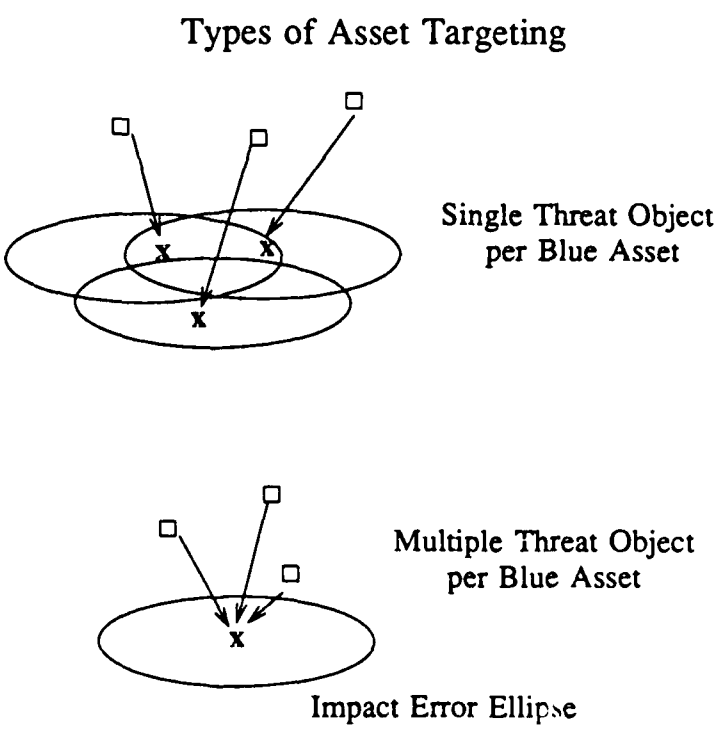


Figure 1. Types of targeting supported by WARM algorithms.

b. We assume that a weapon may be used against TOs aimed at different targets, that multiple weapons may be used against the same TO, that the destruction of one TO is independent of the destruction of another TO, and that the defense knows how many threat objects there are during an attack and the target of each TO.

c. The size of the problem, i.e. the number of weapons times the number of objects, may be very large -- too large for reasonable solution times. It would be advantageous to aggregate some of the objects in order to reduce the number that must be considered by the algorithms. Accordingly, a set of TOs at a given instant occupying a predetermined volume of space (bin), traveling as an ensemble from bin to bin in discretized time steps, and terminating in the same area of the continental US (CONUS) is called a swarm (see Figure 2).

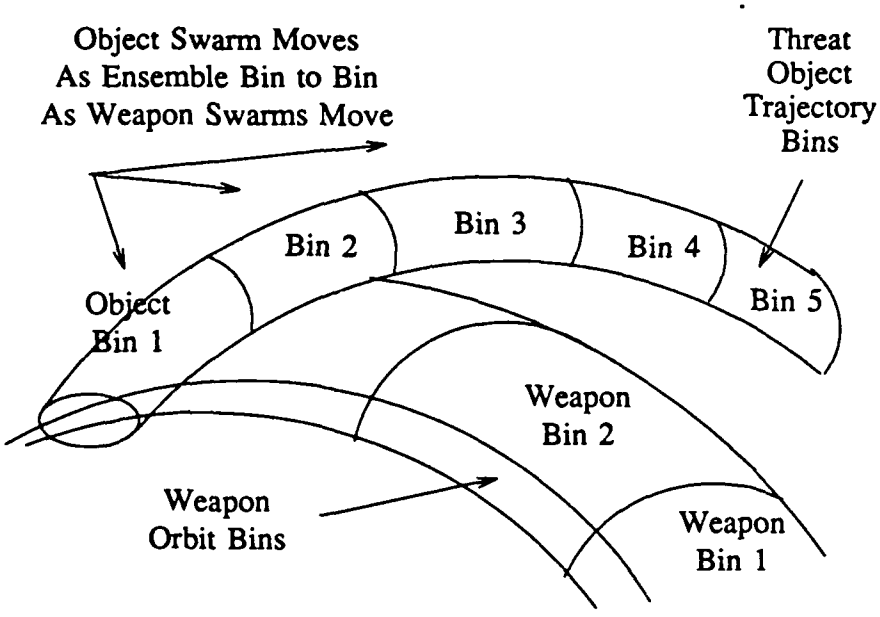


Figure 2. Swarming of weapons and threat objects into spatial bins.

Weapons (and sensors) can also be swarmed in a manner like that of the TOs. Each weapon's (weapon swarm's) shots at a threat swarm (TS) must be allocated fractionally across the constituent objects.

We will refer to a *global* and a *local* problem (See Figure 3). The global problem is concerned with shots from weapon swarms to threat swarms in differing time periods over the battle space time horizon. The local problem takes the global weapon swarm-threat swarm allocation and individual shots from particular weapons to particular TOs in differing firing cycles over the next WARM "think" cycle.

A more detailed mathematical definition of a swarm and a SASP group will be given in Section 2.2.

d. We assume that the defense is able to determine the attack strategy

$$J = \begin{bmatrix} J_1 \\ \cdots \\ J_K \end{bmatrix}$$

where J_k denotes the set of TO indices targeted at k, to be refered to as SASP k. The offensive threats and their targets are known and may not change during an engagement. We define a *firing schedule* α to be a matrix

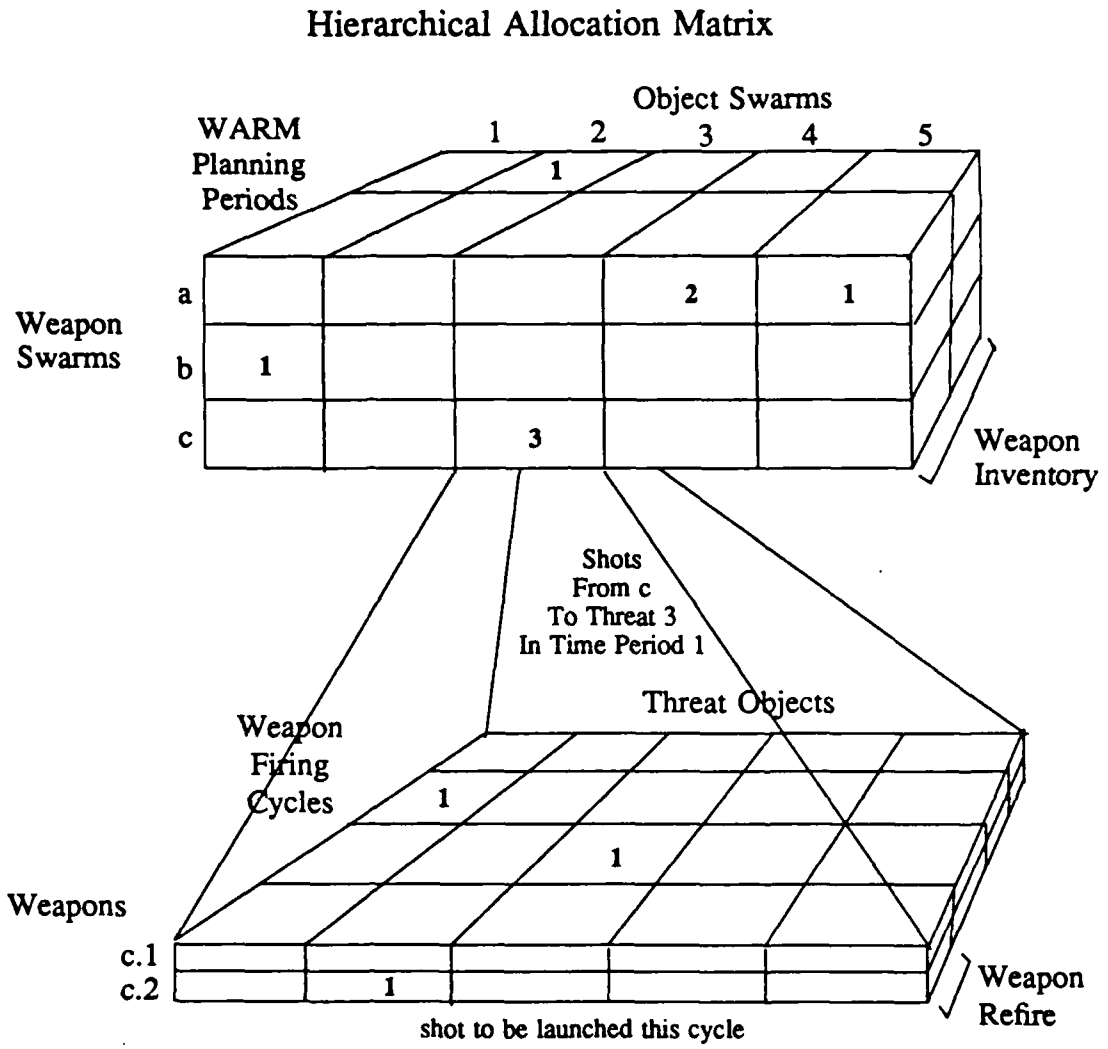


Figure 3. Allocation matrix for global and local weapon allocation problems.

$$\alpha = \begin{bmatrix} \alpha(t_0) \\ \cdots \\ \alpha(t_T) \end{bmatrix}$$

with, e.g., *firing plan* at t_0

$$\alpha(t_0) = \begin{bmatrix} \alpha_{1,1}(t_0) & \cdots & \alpha_{1,N}(t_0) \\ \cdots & \cdots & \cdots \\ \alpha_{1,N}(t_0) & \cdots & \alpha_{M,N}(t_0) \end{bmatrix}$$

where $\alpha_{ij}(t_0)$ is the number of shots from weapon i allocated to TO j in time period 0. We note that the $\alpha_{ij}(t)$ are strictly nonnegative integer values. Shots are fired at specific objects at specific times. Once fired they cannot be redirected. There is an upper bound on the number of shots that can be fired from a weapon. For scheduling purposes time is divided into a number, T, of discrete intervals, $t_0, t_1, \ldots, t_T$ between some initial firing time and a time horizon and there may be a constraint on the number of shots fired from a weapon in a particular time period.

e. Let $p_{ij}(t) \in [0,1]$ denote the probability that shot i fired at time t kills TO j. We assume that the $p_{ij}(t)$'s may be different from weapon to weapon and TO to TO, in fact, many $p_{ij}(t)$'s may be

zero, leading to a sparsely filled Pk matrix (3 dimensional).

We now present five defensive weapons allocation problems, starting with the general problem statement and giving successively simpler (and more tractable) models of damage to defensive assets.

2.1 General Problem

As the TOs become identified with their targets, the value of destroying the object is related to the surviving value of the target at present and in the future. A particular assignment of weapons to objects may entail a whole set of shot-object encounters, each with a probability of successful destruction of the object. Furthermore, any of a SASP group's TOs that leak through cause the same amount of damage, no matter how many leakers there are. The result of all these encounters is a random event -- a set of "leaking" TOs to damage defensive assets. The objective, then, is to find an allocation that maximizes the total expected surviving value of the targets under attack. This objective function can be expressed by the general problem (1a,...,z).

$$S[\alpha^*] = \max_{\alpha_{ij}}(t) \sum_{k \in K} w_k \left\{ \prod_{j \in J_k} Pr \{ \textit{no TO in swarm } j \textit{ survives firing schedule } \alpha \} \right. \quad (1a)$$

$$+ \sum_{m \in J_k} f_k(m) \prod_{\substack{j \in J_k \\ j \neq m}} Pr \{\textit{at least one TO in swarm } m \text{ and } \textit{no TO in swarm } j \textit{ survives}\} \quad (1b)$$

$$+ [\textit{ fractional value when at least one TO in exactly 2 swarms survives } \alpha] \quad (1c)$$

$$\left. + \cdots + f_k(all j \in J_k) \prod_{j \in J_k} Pr \{\textit{at least one TO in swarm } j \textit{ survives firing schedule } \alpha \} \right\} \quad (1z)$$

where

$S[\alpha^*]$: *expected surviving value of optimal allocation* α^*
$f(\,)$: *fractional damage function* $\varepsilon[0,1]$
j : *index of TO or swarm of TOs*
J_k : *set of indices of swarms of TOs going to target k*
k : *index of target sites*
K : *set of indices of sites k to be defended*

The decision variables are the $\alpha_{ij}(t)$s. This formulation permits specific damage outcomes to be identified with each combination of leaker. This formulation is presented for completeness, but is too complicated for practical solution.

Swarming causes several problems with respect to formulating the objective function with SASP grouping. A SASP group may be composed of TOs existing in different swarms due to offensive cross targeting, as illustrated in Figure 4. Swarm P2 is composed of TOs in SASP groups G2, G3 and G4; G4 is split between swarms P2 and P3. When shots are allocated in the global problem to swarm P2, these shots, at the local problem, must be applied such that if #7 is shot at then also must #8 be shot at; there is no benefit to shooting #7 alone with, say #6, with two shots.

2.2 Swarmed Objects, No Partial-Damage Model

The concept of swarms of objects results in two types of swarms: a trivial swarm composed of one object only, and a swarm composed of more than one object. In the former case the swarm would only do damage if the single object was a real reentry vehicle (RV) (the possibility exists of it being a dummy or debris), and if it survives all shots against it. In the latter case the swarm would do damage if one or more of the constituent objects in the swarm was a real RV and survived all shots. The general model distinguishes the effects of real RV leakers from different swarms, and allows for partial damage. Thus in the general model the arguments for the partial damage functions are the specific swarms where at least one constituent object gets through and is a real RV. The probability of the leakage part of this event is assumed independent of that of the real RV part of the event, and so the leakage probability multiplies the probability of being a real RV. The latter probability is actually

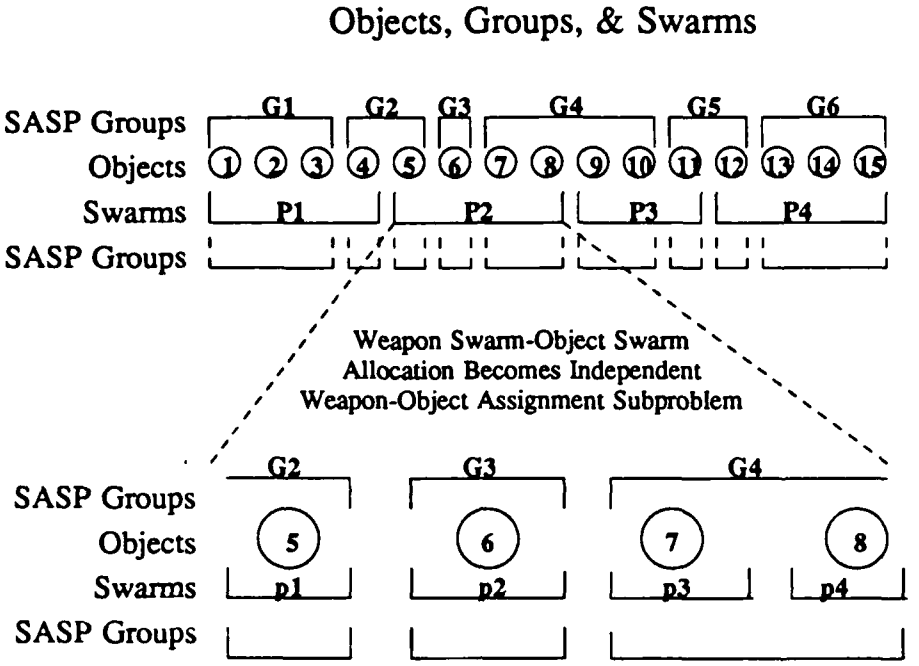


Figure 4. Relationship between threat objects, target groups, and swarms.

incorporated into the partial damage function.

In the more specific model described in this section these probabilities of being real RVs are explicitly stated as coefficients (β). Another feature of this model is the incorporation of the probability that a TO may be destroyed by previous in-transit shots before the shots to be allocated intercept it. This probability is introduced as another coefficient (ξ). Both of these explicitly stated probabilities are independent of each other and of the kill probabilities of the current shots. Furthermore, these probabilities are the same for all members of a swarm. Finally, in this model we assume that any one TO that gets through the defense essentially destroys all of the value of its target -- all of the "partial-damage" f(.) values from the general formulation (1b,...,z) are 0. These assumptions plus some additional constraints on weapon performance and inventory transform the general model to the following form:

$$S[\,\alpha^{*}\,] = \max_{\alpha_{\cdot\cdot\cdot}} \sum_{k} w_k \prod_{s} \left[1 - \prod_{t=0}^{T} \prod_{v} (1 - p_{vst})^{\alpha_{\cdot\cdot\cdot} / \sum_{k} n_{ks}} \right]^{n_{ks}}$$

$$\sum_{t=0}^{T} \sum_{s} \sum_{v} \alpha_{vst} \le M \; : \; \textit{imposed limit on total shots fired}$$

$$\sum_{t=0}^{T} \sum_{s} \alpha_{vst} \le I_v \; : \; \textit{inventory of weapon swarm } v$$

$$\sum_{v} \sum_{s} \alpha_{vst} \le B_t \; : \; \textit{imposed budget of shots in time } t$$

$$\sum_{s} \alpha_{vst} \le R_{vt} \; : \; \textit{limit on shots from weapon swarm } v \textit{ in time } t$$

$$\alpha_{vst} \in Z^{+}, \; \textit{nonneg. integers}$$

k : *the index over groups*
w_k : *the value associated with group k*
s : *index over threat object swarms*
v : *index over weapon swarms*
T : *time period at end of battle*
t : *index over time periods*
p_{vst} : *probability a shot from swarm v at time t gets a member of s*
α_{vst} : *number of shots from weapon swarm v at time t assigned to s*
n_{ks} : *number of objects in the intersection of group k* and *swarm s*

For notational simplicity we refer to the time intervals $t_0, ..., t_T$ with the index $t\varepsilon 0,T$.

2.3 No Swarms, No Partial-Damage Problem

When we assume that each object is distinguishable (one per swarm) and that any one TO that gets through the defense essentially destroys all of the value of its target, we have (3).

$$S[\alpha^*] = \max_{\alpha_{ijt}} \sum_{k\varepsilon K} w_k \left\{ \prod_{j\varepsilon J_k} \left[1 - \beta_j \xi_j \prod_{t=0}^{T} \prod_{i\varepsilon I} (1 - p_{ijt})^{\alpha_{ijt}} \right] \right\} \tag{3}$$

subject to (same as (2))

This problem probably occurs when precise target/TO pairings are known and may occur during the final stages of an engagement. In this case one leaker causes total damage of the target.

2.4 One Threat per Target Problem

If we make the further assumption that there is at most one TO aimed at each target, or each target is "small" enough that each TO's damage can be considered independent of all others, then the product over $j\varepsilon J_k$ of (3) is reduced to one term. Now the objective is to minimize the inner product, which is the expected damage to the targets from "leakers" through our defense when we make our set of $\alpha_{ij}(t)$ assignments. Since now the j's and the k's are synonymous, we retain the j index instead of k.

$$S[\alpha^*] = \min_{\alpha_{ijt}} \sum_{j=1}^{N} w_j \beta_j \xi_j \prod_{t=0}^{T} \prod_{i\varepsilon I} (1 - p_{ijt})^{\alpha_{ijt}} \tag{4}$$

subject to (same as (2))

This objective is most applicable when precise targeting is not known and potential target values can be aggregated across an area target. It is also applicable when object damages are small enough to be independent of one another.

2.5 Single Shot per Threat Problem

If in (4) each TO j is restricted to a single shot during the firing regime $[t_0,t_T]$ and each weapon can only fire once during any time interval t then the problem assumes it simplest form

$$S[\alpha^*] = \max_{\alpha_{ijt}} \sum_{j=1}^{N} \sum_{t=0}^{T} \sum_{i\varepsilon I} w_j \beta_j \xi_j p_{ijt} \alpha_{ijt} \tag{5}$$

subject to

$$\sum_{t=0}^{T} \sum_{i\varepsilon I} \sum_{j=1}^{N} \alpha_{ijt} \le M \le \sum_{i\varepsilon I} I_i \text{ : total of } M \text{ shots} \le \text{total inventory} \tag{5a}$$

$$\sum_{t=0}^{T} \sum_{i\varepsilon I} \alpha_{ijt} \le 1 \text{ : one shot for each TO } j$$

$$\sum_{j=1}^{N} \alpha_{ijt} \le 1 \text{ : one shot by each weapon in any } t$$

$$\alpha_{ijt} \varepsilon \{0,1\}$$

This objective is applicable with a "shoot-look-shoot" firing strategy using kill assessment feedback. The battle manager shoots one (or more) shots, "looks" to determine a kill or miss, shoots again if it missed, etc. In this formulation the time intervals t might correspond to weapon refire times (thus

allowing only one shot per weapon per time period) and the regime T might correspond to the time for intercept and kill assessment. Using this strategy, the battle manager has good shots, good kill assessment, and has battle space to fire shots sequentially one at a time to reduce wastage.

We now explore some methods which might solve these objectives; the methods will be drawn from the areas of mathematics, operations research, and artificial intelligence.

3. CLASSICAL METHODS APPLIED TO NP-COMPLETE PROBLEMS

The next section presents an overview of classical techniques, spelling out their use in solving problems (2), (3), (4), and (5). The subsequent section presents a detailed description of some candidate algorithms for our algorithm suite and how they might be integrated to work together.

3.1 Classical Techniques

Mathematical models in operations research may be viewed generally as determining the values of the decision variables x_j, j ε {0,1,2,...,n} which

$$\begin{aligned} &\textit{optimize} && x_0 = f(x_1, x_2, \ldots, x_n) \\ &\text{subject to} \\ &&& g_i(x_1, \ldots, x_n) \le b_i,\ i = 1, 2, \ldots, m \\ &&& x_j \ge 0,\ j = 1, 2, \ldots, n \end{aligned}$$

The function f is the objective function, while $g_i \le b_i$ represents the i^{th} constraint, where b_i is a known constant. The constraints $x_j \ge 0$ are called the nonnegativity constraints, which restrict the variables to zero or positive values only.

3.1.1 Linear Methods If f and g_i are linear functions of the decision variables x_j then the problem may be formulated as a linear program LP

$$\begin{aligned} &\textit{Maximize} && x_0 = \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} \\ &&& \sum_{j=1}^{n} a_{ij} x_j \le b_i,\ i = 1, 2, \ldots, m \\ &&& x_j \ge 0,\ j = 1, 2, \ldots, n \end{aligned}$$

Only formulation (5) qualifies as a linear program. In particular, if the shot limit and inventory constraint (4a) is removed or strict equality holds, the problem becomes a special and easily solved case of an LP called an "assignment" problem. One of the fastest assignment algorithms is by Gabow and Tarjan [3] with run times roughly proportional to $(n^{1/2})*m*logn$ where $n=(N+M)$ and $m=N*M+n$. When strict inequality of (5a) holds, the assignment algorithm is not valid and a (0,1) integer LP will solve problems of relatively small size. Only (5) is linear, problems (2), (3), and (4) are nonlinear.

3.1.2 Nonlinear Methods Classical optimization theory involves the use of differential calculus to determine points of maxima and minima (extrema) for unconstrained and constrained nonlinear functions. The methods may or may not be suitable for efficient numerical computations. In general consider the problem

$$\begin{aligned} &\textit{maximize} && x_0 = f(\mathbf{X}) \\ &\text{subject to} \\ &&& \mathbf{g}(\mathbf{X}) \le \mathbf{0} \end{aligned}$$

where f and $\mathbf{g}$ are nonlinear function(s) of the decision vector $\mathbf{X}$.

The *Kuhn-Tucker* conditions are sufficient for any local maximum (minimum) to be a global maximum (minimum) if both f and the $\mathbf{g}$ are concave (convex). Lloyd and Witsenhausen [4] have shown that although (4) is concave, it unfortunately is NP-Complete[1], and thus cannot be solved for large sized

problems. One approach that has some hope of finding the global maximum to (4) is to find a maximum by some nonlinear method and then to employ some rounding scheme to get the integer solution. Concavity (convexity) is useful in this situation in that it provides an upper (lower) bound on the integer solution; such information could be used to determine whether or not to try to get a better (than current) solution by some rounding schemes. Note that we insist on integer solutions, and thus any solution based on derivatives, i.e., real solutions, must at last be rounded by some scheme. To our knowledge no generalized rounding scheme exists to yield the optimal integer solution to an optimal continuous solution of a concave (convex) NLP.

Unfortunately problems (2) and (3) are neither strictly concave nor strictly convex. Although a local maximum may be found, the Kuhn-Tucker conditions are no longer sufficient for this maximum to be the global maximum. A simple example shows why (2) and (3) are multimodal [5],

$$f(x,y) = (1-e^{-x})(1-e^{-y}) \text{ for } x,y \geq 0$$

The separate factors of f are concave, but their product is not concave; the Hessian is explicitly

$$\begin{vmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{vmatrix} = e^{-(x+y)} \left[1 - e^{-x} - e^{-y} \right]$$

This is negative for x,y near the origin (concavity condition) but is positive for larger x,y (convexity condition). A maximal solution to a concave NLP for (2) and (3) tells us nothing about the global maximum; and we still have the problem of rounding to an integer solution.

With a multimodal objective function and integer decision variables we have a problem in combinatorial optimization. We cannot say that (2) and (3) are NP-Complete and can only conjecture that surely they are NP-hard.

3.2 Other Methods

Other methods for solving discrete optimization problems fall into two basic classes whether the method originated in mathematics, operations research, or artificial intelligence. The two classes are deterministic algorithms and randomized algorithms. We will restrict our presentation of algorithms according to their applicability to problems in combinatorial optimization - the study of solutions to NP-hard problems.

3.2.1 Deterministic Algorithms Perhaps the most popular method for discrete problems is *dynamic programming DP*. However, DP is plagued by the *curse of dimensionality*, the number of states can become very large. There is little hope that DP can solve combinatorial optimization problems when DP itself is hampered by combinatorially large state spaces for relatively small problems.

Two other deterministic methods are the *Branch & Bound* and the *Greedy* methods. These methods are explored in much greater detail for our use in their own sections, 4.2.1 and 4.1.3 respectively.

3.2.2 Randomized Algorithms Existence of NP-hardness is enough circumstantial evidence that no algorithm of polynomial time is likely to be found. [6] Randomized algorithms, i.e., algorithms with randomized operations, rule selection, or steps, offer the possibility of globally exploring the feasibility region and avoiding being trapped at a local optimum. The price is computation time. There are numerous randomized algorithms for combinatorial optimization, and a few of the more promising ones are detailed in separate sections below. (At present we know of no randomized algorithm that has been

1. A problem belongs to the class of NP-Complete problems if the optimum cannot be *known*, short of exhaustive enumeration, and any algorithm solving this problem will, by some transform, solve all problems in the class NP-Complete. NP-Complete problems belong to the set of NP-hard problems which are characterized by algorithms with run times bounded by an exponential function of the problem size. P problems belong to the set of NP problems but are characterized by algorithms with run times bounded by a polynomial function of the problem size.

characterized to be either polynomial-time or non-polynomial time.)

All randomized algorithms share a common difficulty - convergence criteria and stopping rules. Four approaches to defining convergence have emerged:

a. Empirical analysis compares one heuristic against other heuristics on the dimensions of computing time and near-optimality. One heuristic dominates another if both near-optimality *and* computing time are better, all other things being equal. Parameters controlling rate of convergence and quality of solution are tuned according to results. It becomes difficult to transfer results from one situation or problem to another. This is a very case dependent and therefore unsatisfactory, albeit expedient, method of determining convergence criteria.

b. Probabilistic analysis requires that a probability distribution over the set of problem solutions be specified.

$$Prob\left\{\frac{Heuristic\ solution\ value}{Best\ solution\ value} \le 1-\varepsilon\right\} \le 1-\delta$$

where for any $\varepsilon>0$, $\delta\rightarrow 0$ as the number of samples goes to ∞. According to the cumulative distribution function, we can get arbitrarily close to the maximum with probability arbitrarily close to unity. To form a probabilistic stopping rule one would specify an ε and a δ and sample solutions until the largest sample value satisfied the above probabilistic statement. The essential difficulty for this approach is coming up with the distribution of solutions. Note that the maximal solution value is part of this distribution although the allocation yielding this maximal solution is not known. Bounding the "Best solution value" from above is not appropriate since, if the bounding error is greater than ε, the randomized algorithm would never stop. Bounding from below encourages stopping, albeit earlier than preferred.

c. Statistical analysis relies on results from extreme value theory. The distribution of extreme values (the distribution of the largest (smallest) in a sample of very large size) is a Weibull where

$$F(x_0) = Prob\{x \ge x_0\} = 1 - \exp -((x_0-a)/b)^c$$

with $a \ge x_0 > 0$ and $b,c>0$ as scale and shape parameters. To form a statistical stopping rule one would define a "lot size" of sample solutions, retain the largest solution in each lot (to get the extreme values), use maximum likelihood estimates of b and c from the extreme values, and, knowing x_0 (the most extreme sample solution) and guessing at a (the global maximum), iterate until satisfying the "tail" probability $F(x_0)$.[7] This approach overcomes the difficulty of knowning the probability distribution of solutions, but still has the difficulty of providing the "guess" a of the global maximum. Furthermore, to be theoretically valid, this method requires a very large number of *statistically independent* samples.

d. Bayesian analysis fits a decision-theoretic framework to the results of various trials and, for each new trial(s), estimates the likelihood of obtaining a significant improvement, i.e., in general if $P(x \ge x_0) < C_S / C_O$ then stop computing, where C_S is the cost of additional samples (computer time) and C_O is the expected cost of not having the better answer. There is no pretense at estimating the global optimum nor of divining the distribution of solutions as in the previous approaches.

With respect to these stopping rules, two randomized search programs are currently popular, Simulated Annealing and Genetic Search. Simulated Annealing is similar to the process of "cooling" where the lowest entropy point is the "optimal" in the case of minimization. This point is found by randomly modulating the solution vector according to entropy laws in the "direction" of lowest descent; sometimes the random direction is upwards which gets over "little local minimums". Whereas Simulated Annealing works on only one solution vector at a time, the Genetic Search mates pairs of solutions in much the same way as the genetic process combines DNA strings and encourages "survival of the fittest". The Genetic Search has very well defined rules, statistically and theoretically based in genetics, for making candidate solution vectors whereas the Simulated Annealing rules are heuristics without theoretical basis. For this reason we have chosen to implement the Genetic Search as an algorithm of randomization. The

stopping rules suggested by Holland although theoretically sound are not of much practical value for the problem of asset selection. We have investigated several approaches for stopping rules for the Genetic Search and our implementation is reported in Section 4.1.3.

4. AN ALGORITHM SUITE FOR WEAPONS ALLOCATION

It was shown in Section 3 that the inclusion of the weapon assignment problem in the class NP-Complete poses some difficulties for a solution technique. These difficulties are compounded by the potentially immense size of the problem (100,000 objects and 10,000 weapons). Accordingly, an overall solution strategy should include, in addition to heuristic algorithms for dealing with the combinatorial optimization inherent in (2), (3), (4), and (5), techniques for reducing the size of the problem and possibly breaking it into subproblems for separate solution. One technique for problem size reduction is to aggregate objects first, and consider only the aggregated entities (swarms) in the subsequent allocation. Because the allocation problem size is a function of the total number of combinations of weapons times entities allocated to, the effect of a prior reduction in the number of entities is multiplied many times.

Besides the difficulties inherent in the problem size, there is also the difficulty of redundantly targeted assets. Because of the very strong implications of the assumption that all incoming RVs must be destroyed for an asset to survive, the allocation problem decomposes naturally into two subproblems: a selection of a set of assets to defend, and an allocation of shots to the objects aimed at the defended assets. Treating the problem as two subproblems, with separate algorithms addressed to each subproblem but acting iteratively to achieve an overall solution, results in additional benefits of flexibility. It will likely not be the case that a single or small set of overall weapon allocation algorithms can handle well the wide range of circumstances and scenarios that could be seen. The flexibility of allowing any of the set of asset selection techniques to operate in concert with any of the weapons allocation schemes increases considerably the variety of overall allocation methods, and will tend to extend the range of situations that can be handled well.

We have implemented both swarming and the decomposition of the problem into two iteratively solved subproblems. The order that these functions are performed, as well as other characteristics of the problem, affect the overall running time. For example, if asset selection is done first, followed by swarming and finally weapons allocation, the swarming effort, which is certainly dependent on the number of objects to aggregate, would only have to treat those objects going to the selected assets (thus saving computer time). On the other hand, swarming within the asset selection/weapons assignment iterative loop may entail very many redundant swarmings of objects that continually reappear in the set of those going to defended assets. The balance between these two effects depends, in part, on the fraction of assets defended (and thus the fraction of objects to be swarmed). If the number of shots available is small compared to the number of objects, and if the number of iterations is small, the benefits of including swarming within the iteration loop may outweigh the detriments. It was our judgement, though, that this would not be the case in general, so we implemented these two functions with swarming preceding the asset selection/ weapons allocation iterative loop.

We present the techniques for implementing the swarming of threat objects and weapons (to be extended to sensors) in a report on our internal testbed LETS, since, these algorithms are completely independent of the weapon allocation problem. Within the context of swarming, though, we note that a swarm of weapons may be allocated shots directed toward a swarm of threat objects, just as shots may be allocated from individual weapons toward individual threat objects. We will try to present differences between weapons and swarms of weapons when relevant, but the reader should always consider a weapon to be synonymous with weapon swarm, and likewise with threat objects.

The descriptions herein concentrate on the algorithms for solving the asset selection/weapons allocation subproblems. The two sets of algorithms from which a pair of coordinating algorithms is chosen are collectively called the algorithm suite. The selected pair from the algorithm suite iteratively computes a solution in two stages. At the first stage a set of assets are proffered for defense to the second stage, which, given the set of assets to defend, invokes a local search algorithm for allocation of weapons to TOs. After the local search returns a solution another set of assets is chosen for defense. These stages

are repeated iteratively. The best solution at the end of a decision cycle is delivered to SASP and to the weapons.

The heuristic algorithms presented here for solving problems (2), (3), (4), and (5) are based on the discrete deterministic and randomized optimization methods, described in Section 3, and exploit the features of the various problems.

In selecting among the algorithms particular attention is paid to those algorithms which iteratively build a solution of higher quality from a previous solution. (and consequently will have longer run times with more iterations). The idea behind this approach is to obtain one solution as soon as possible, and with more compute time, to get a better and better solution.

A three *asset selection* candidate algorithms are described. Each may invoke any one of three *weapon allocation* algorithms. The possible combinations of asset selection and weapon allocations algorithms are:

Weapon Allocation Algorithms	Asset Selection Algorithms		
	King-of-Hill	Best Branch	Genetic
ALIAS Greedy	X	X	X
PAIRswap Greedy	X	X	X
MARGinal Greedy	X	X	X

Each of the 9 combinations will undergo testing to determine its best operating conditions. These methods are the only ones included in the WARM software, although other algorithms were considered during the life of the contract.

4.1 Asset Selection Algorithms

Because an asset must be defended by at least as many shots as there are objects aimed at it in (2) and (3), total defense of all assets may not be feasible or cost effective. Therefore it is important, as part of the allocation algorithm, to determine what assets will be defended, and which will be left undefended. The weights assigned by SASP to the assets affect the asset defense decisions. For K assets each to be defended or undefended, there are 2^K possible combinations when M (#shots) $\geq N$ (# TOs), otherwise there are less than 2^K. Exhaustive search may not be practical and a randomized algorithm (e.g., Genetic search) or heuristics must be employed. The following sections describe in detail the asset selection algorithms included in LETS.

4.1.1 King-of-the-Hill Much like the game of King-of-the-Hill where K subjects scramble for the top of the hill and the last one on top is eliminated at each "round", this asset selection heuristic eliminates, at each iteration, the asset with the poorest shot utilization until the objective function "peaks".

The algorithm proceeds by the following steps:

i. Initially consider all assets to be defensible, set n $\leftarrow$ 0 (the number of assets eliminated).

ii. Determine a weapon allocation via some method discussed in 4.2.

iii. Knowing the optimal shot allocation α^* determine the asset, k_n^*, that is hardest to defend by

$$g(k_n^*) = \min_{k \in K} \left\{ \frac{\sum_{j \in J_k} w_j \left[1 - \prod_{t=0}^{T} \prod_{i \in I} (1 - p_{ij})^{a_{ij}^*(t)} \right]}{\left(\sum_{t=0}^{T} \sum_{i \in I} \sum_{j \in J_k} a_{ij}^*(t) \right)} \right\}, \qquad (6)$$

k_n^* is the asset with the lowest expected surviving value per shot expended on its behalf. Drop this asset from the defensible set and set $n \leftarrow n+1$.

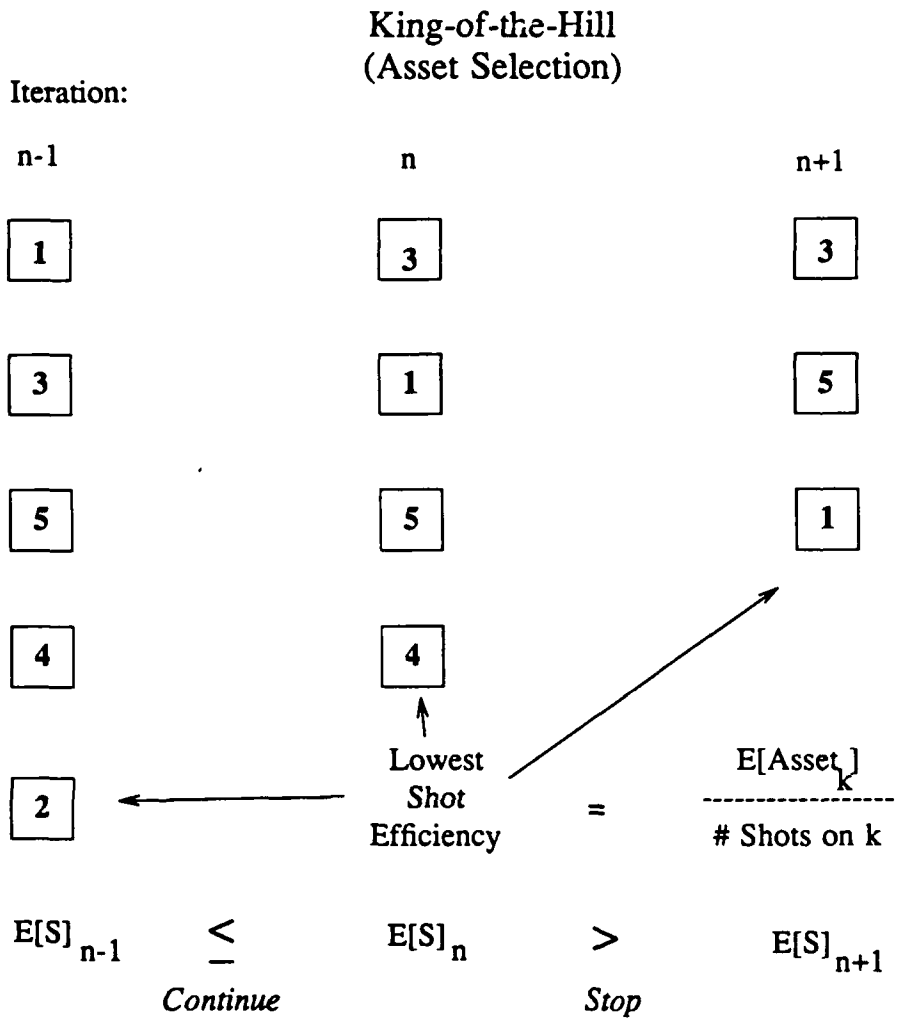


Figure 5. King-of-Hill drops the least defendable asset until a maximum is found.

Equation (6) determines the criterion by which sites become undefended to free up shots for use in defending "more important" assets. This criterion is based on expected surviving value per shot, or the average killed TO value per shot for each asset. The asset dropped from the assets to be defended is that whose ultimate value, w_k, seems to be hardest to achieve. There are two cases to be considered. One occurs when $M<N$; a cover cannot be obtained. In this case, some assets will be undefended because some TOs are coming through. But, we have assigned (on a single shot basis) the best possible shots and the asset with lowest expected value per shot must be deemed the hardest to defend. Thus, drop it, and use its shots to defend higher expected values. In the second case, $M \geq N$, a cover over the selected assets is possible. The allocation can be determined, and again, the hardest to defend asset is dropped.

An interesting situation occurs when, for example, one asset has at least one shot on each of the objects aimed at it (asset covered) and another asset has at least one object with no shots (asset uncovered). The shot efficiency measure for the covered asset is the lowest, lower than the shot efficiency measure for the uncovered asset. Should the covered asset be dropped and the uncovered asset retained? Many trials substantiate and affirmative answer; the explanation is that the uncovered asset has a large weight and the covered asset has a small weight and high Pk shots. Care must be taken to see that no uncovered assets remain in the final solution even though the objective function may "peak". Such a rule can cause the expected surviving value to fluctuate

as asset are dropped one at a time. This fluctuation will be described further later in this section.

iv. Repeat steps ii-iii until the objective function "peaks" (maximized) *and* when all assets have either been eliminated or are in the solution; or all assets except the one in the solution have been eliminated. That is,

$$S_n(\alpha_n^*) > S_{n+1}(\alpha_{n+1}^*),$$

the $(n+1)^{th}$ asset deletion reduces expected surviving value from the n^{th}. As the hardest to defend assets are dropped, one by one, the shots become more effectively used, the value of $S(\alpha^*)$ increases until increased shot efficiences cannot offset the value of an undefended asset and $S(\alpha^*)$ begins to fall.

This heuristic is intended to be fast, with $(K-1)$ iterations in the worst case, and does not attempt to reconsider dropped assets to seek better combinations of defensible assets. Figure 5 shows how the ordering of assets from iteration to iteration might change as a function of reallocated shots. At iteration $n-1$, asset #1 is the most defendable while asset #s is the least defendable; asset #2 is dropped and its shots are reallocated to the defense of assets #1 and #3-5. Apparently those very shots must have been applied toward the defense of #3 since it is now the easiest to defend. The expected surviving value of the defense has increased by dropping #2 and reapplying its shots elsewhere. For the next iteration, asset #4 is dropped and the strategic shot limit is reallocated across the defense of the remaining three assets. In this iteration, unfortunately, the expected surviving value of the defense *decreases;* that is, the marginal value of the reuse of the shots on #4 is negative.

We have tested this heuristic against several very large scenarios, approximately 1000 threat objects and 180 assets. Some refinements to the basic King-of-the-Hill algorithm have been required, due to the fluctuations in the values of the objective function, as mentioned above. Instead of stopping after a "peak" of just one dropped asset, we now look for a "peak" of 90%. That is, we now define a peak in the objective function to have formed if the current objective function value is less than 90% of the incumbent solution (the best solution with no uncovered assets). Assets are dropped one at a time after a peak is found until only 90% of the best solution remains, and then King-of-the-Hill stops and returns the incumbent solution.

A slight twist on this rule is the following. We try all combinations of the first two assets to be dropped just after a peak is found. This might be generalized to the first n assets, but we have not implemented such a version. For example, asset A is dropped and the objective function decreases, but is still greater than 90% of maximum. Next A is restored and B (computed on the basis missing A) is dropped. If the objective function increases, continue with A in and B out, dropping C next. If the objective function decreases, drop A also. Sometimes, the objective function will increase due to the redistribution of shots from A and B. If on the other hand, the objective function decreases, continue dropping assets until either the objective function again peaks indicating a prior "local" peak, or the 90% limit is attained and search is stopped.

4.1.2 Best Branch The Branch and Bound (B&B) algorithm is another standard deterministic tool used in integer program optimization. B&B builds a tree of alternative solutions and maximizes the objective function by searching the tree and pruning branches whose upper bound on the objective function value is less than some solution already in hand. (An upper bound of a branch is a value not less than any possible solution from the branch.) In the weapons allocation case, B&B would work by considering, in turn, the abandonment of each asset defended in the current allocation. (These are the branches from the current solution.) An upper bound on the objective function resulting from the abandonment of an asset is compared to the best solution found so far; if it is not as good, the abandonment is not considered. This technique achieves efficiencies if these upper bounds are both tight and quickly determinable -- then there is the possibility of quickly pruning many branches of the search tree so that they don't have to be searched.

We have adapted a slight variation of the B&B approach we call the Best Branch (BB) algorithm: all branches of the current solution are considered sequentially, a pseudo upper bound is found for each, and we prune all but the best branch, stopping if the best branch from a current solution is not as good as the current solution. The "upper bounds" are in fact the complete solution values when an asset is

not defended. If our algorithms always produced optimal solutions at a branch these would be true (and tight) upper bounds for the solutions at all subbranches, but because our algorithms are instead heuristics we do not have true upper bounds and thus cannot call this a true B&B technique. Since our solution algorithms are heuristics, we may achieve a solution value which is less than the actual optimum and thus we could prune a branch erroneously. Unfortunately, there are no good upper bounding techniques that are both quick and tight, and loose upper bounds -- those far above the actual bounds -- would tend to reduce pruning and cause a tremendous amount of extra branch searching. Given that the Best Branch variation eliminates the these time-consuming searches, it is a reasonable alternative to the Branch and Bound technique.

A description of the overall BB algorithm follows:

1. Initially, find the best allocation when no targets are abandoned. Call the solution value the best current solution.
2. Temporarily abandon the next asset not abandoned in the current solution.
3. Use the shot allocation schemes to determine an allocation of shots to TOs. However, do not allocate any shots to TOs attacking targets that are abandoned. Obtain the objective function value for the allocation.
4. Restore the temporarily abandoned asset to the defended category. If there is no next asset to be abandoned, go to (5), else return to (2).
5. Determine which asset's abandonment leads to the best objective function value. If this value is less than the best current solution, STOP, the best current solution is the final solution. Otherwise make the asset permanently abandoned and its solution value the best current solution. Go to (2).

4.1.3 Genetic Search The genetic search algorithm was first introduced by J.H. Holland [8] in 1975 and is currently popularized as a technique for searching an expert's knowledge base and a related area of data base retrieval.[9] Its difficulty in gaining acceptance in a more generalized format for search is the formulation of a "string" somehow packaging all the information required to evaluate optimality and simultaneously permitting the application of genetic operators. Genetic search has been studied for function optimization.[10] Most recently genetic search is playing a role in machine learning where in goal-seeking a rule base is generated and modified according to experiential information by genetic operators. [11]

4.1.3.1 Theoretical Development A full justification for the efficiency of the Genetic Search cannot be given in this report and the reader is referred to our references for greater detail. The following presentation has been adapted to provide a more useful exposition of the search technique for asset defense selection.

A combinatorial optimization problem may be stated in the following way: "Given a family P of feasible combinatorial structures, each represented by a point p, and a utility $u(p)$ defined over the members of P, find the one of them with maximum (minimum) utility." For each point p we assume that a neighborhood $N(p)$ is defined by other points close to p (by some metric) and will define $N(p)$ in such a way that an operator (e.g. a "cross-over") acting on two points in $N(p)$ will give another point (i.e. an "offspring") in $N(p)$. Furthermore points for the operator will be selected from $N(p)$ according to their utilities, $u(p)$, and the result of the operator will be an "offspring" with either a higher or a lower utility. For a maximization problem, only the higher utilities are retained from each generation to form the "offspring" pool from which to make the next generation. The higher the utility, the more often (relative to lower utilities) will point p be selected, and thus there will be more "offspring" of generally higher utility. This process is analogous to "hillclimbing" in classical gradient techniques. The expected number of "offspring" from point $p_{i,n}$ (the i^{th} member of generation n) is

$$O(p) = \frac{u(p_{i,n})}{(1/N)\sum_{i}^{N} u(p_{i,n})}$$

and is greater than 1 if the $u(p_{i,n})$ is greater than the average and less than 1 if less. Thus the best

performer will reproduce at a compounding rate across generations.

Each "generation" of points $p_{\cdot,n}$ creates the next generation of offspring $p_{\cdot,n+1}$ via a set of genetic operators. The *cross-over* genetic operator is the primary operator and mates two asset selections to yield two new asset selections. Mate selection for cross-over occurs randomly by utility, according to the selection probabilities $u(p_{i,n})/\sum_{i}^{N} u(p_{i,n})$. Consider each asset selection (a vector of 0's and 1's indexed by asset) to be a "string" with a head and tail. A common point for division between the head and tail may be determined randomly or by some feasibility rules. A cross-over between two asset selections occurs by exchanging tails to give two offspring asset selections. For example, the following cross over occurs with a division between positions 5 and 6.

$$1\,0\,0\,0\,1\,1\,1\,0 = 1\,0\,0\,0\,1\,1\,1\,1$$
$$\text{X}$$
$$1\,0\,0\,0\,0\,1\,1\,1 = 1\,0\,0\,0\,0\,1\,1\,0$$

The *inversion* genetic operator occurs after cross-over on a randomly selected segment on an offspring asset selection. The inversion operation occurs with probability δ_n. An example from the preceding cross-over is

$$1\,0\,0\,0\,1\,1\,1\,1 = 1\,0\,0\,1\,0\,1\,1\,1$$

where positions 4 and 5 have been flip-flopped. An inversion operator may take the search from one $N(p)$ to a neighboring $N(p)$.

The *mutation* genetic operator alters a single element after an inversion or cross-over operation with probability γ_n. An example from the preceding inversion is

$$1\,0\,0\,1\,0\,1\,1\,1 = 1\,0\,0\,1\,0\,1\,0\,1$$

where the 1 in position 7 has been changed to a 0. A mutation operator may take the search from one $N(p)$ to some other neighborhood. The operations of inversion and mutation occur rarely and occur principally to prevent the search from getting stuck at local optima. In this way a new generation of feasible asset selections is drawn genetically from the current generation.

4.1.3.2 Application to Asset Selection There are a number of ways of obtaining a "first generation" of asset selections. An attraction of the genetic algorithm is the ability to include as first generation elements those solutions to (2) or (3) computed by other (earlier) algorithms, e.g., the asset selection from King-of-the-Hill (4.1.1) with a random draw to fill out the first generation or Best Branch (4.1.3) which can fully supply the first generation.

As better and better solutions are retained (and "weaker" ones fall away), the survival rate of successive offspring will become lower and lower as champions abound. The probabilities of the mutant γ_n and inversion δ_n operations will determine how quickly the process will converge and how often the search terminates at a local maximum.

There are a few inherent difficulties with the Genetic Search algorithm. (Incidentally, these difficulties also occur in the Simulated Annealing algorithm and to a larger extent). Firstly, there is the difficulty of dealing with "recurring candidates", i.e., the result of the application of the genetic operators will yield a candidate solution which has been tested in the past. This inefficiency of course grows with the length of run time. We attempt to minimize its effect by an efficient test of the candidate against the current pool of champions, but have no test against candidates not in the pool. We actually use the number of rejected candidates as part of a stopping rule. Holland recommends a stopping rule based upon the convergence of the average score of the champion pool to within a given relative range. We have found that this stopping rule is not appropriate for our problem because of the two levels of allocation; the stopping iteration is *very sensitive* to the range of convergence and in fact is problem dependent. If the the objective function is flat near the "peak", then the search converges quickly (depending on the size of the pool). A small pool size and flat objective function converges more quickly than a larger pool. This behavior is not consistent with a robust criterion for a stopping rule. Likewise if the objective function is steep around the peak, then the convergence range may not be met because the size of the

pool is such that the average score would *never* fall within the range.

With regard to the four approaches to a stopping rule presented in Section 3.2.2, we have formulated a stopping rule most akin to the Bayesian approach. In this case we look for the probability of a new incumbent with the very next sample to be less than the incremental relative cost of having sampled since the last champion was found. That is, one can sample from a dry well only so long before drawing the correct conclusion. So we have $P(x \geq x_0) < C_S / C_O$ where x_0 is the present incumbent solution, C_S / C_O = number of duplicates generated since last champion / 3 * number of assets. This stopping rule captures the idea that the search has probably found the best to be found once the genetic operators have regenerated current champions.

In the delivered version of the Genetic Search, the pool of champions is initially populated by random variation of strings of binary values for the defense of the assets. This approach certainly assures that the space of possible subsets of assets is spanned, while "seeding" the pool with the ALIAS solution. We have found this approach to yield satisfactory results. We note that the evaluation of a candidate asset defense vector is very time consuming since a complete weapon allocation must be performed; therefore for weapon allocation algorithms other than the ALIAS, we first do ALIAS, and if its solution is within 95% of the incumbent, only then do we use PAIRS or MARG. This process does not waste computer time in optimizing an inferior solution, even though, after the genetic crossover operation, it may be one mate directly leading to the optimal solution. The genetic search is inherently a parallel algorithm and is included in our algorithm suite for problems where K is large in comparison to M and N. With proper programming and n processors, the Genetic Search should run in very nearly $1/n$ the run time of a single processor computer.

4.2 Weapon Allocation Algorithms

Having determined which assets to defend at a given iteration, it becomes necessary to determine the best firing schedule for M shots against the selected TOs $\leq N$. The following section describes algorithms which would be used for performing the weapon to TO assets selection given a set of assets to defend.

4.2.1 The Greedy Algorithm The greedy algorithm is often used in combinatorial heuristics. It is based on the idea that the solution ought to be found in a finite series of iterations, and at each iteration the best values, according to some metric, of a subset of the decision variables are chosen from the range of possible alternatives. The selection at a particular iteration may affect the choices in future iterations, but these effects are expressly not considered in the current iteration. For example, this technique might proceed by entering into the solution at each iteration, that one shot from a weapon with the largest expected value. A randomized version works by selecting the element to enter the solution at a given iteration randomly according to a probability mass function which attributes higher probability to elements of larger expected value. A Mixed-Random Greedy would perform elements of both, some deterministic and some random. The procedure stops when all shots have been expended.

Worst case performance bounds on the quality of solution for the greedy have been analyzed by Hausmann, et. al., [12] Specific bounds are very problem dependent and they did not study our problem, however, for cases they did study, the bounds for the worst case performance of the greedy was no less than 50% of the true maximum. This result is reassuring but not particularly enlightening. In any case they show that the "k-Greedy"[2] may perform more poorly than the "1-Greedy" in the worst case. Hence, we have selected the 1-Greedy, or the one-shot-at-a-time search.

4.2.2 Greedy Approximation -- The ALIAS Algorithm Another common technique in optimization is to approximate a nonlinear function by piecewise linear functions and solve a sequence of LP problems (separable nonlinear programming). This might apply in the case of the concave (4) if fractional shots were permissible, but unfortunately the integer LP problem is NP-Complete.[13] and hence such an

2. The k-Greedy selects k elements to enter the solution at a given iteration while the 1-Greedy sets k=1.

approximation approach is irrelevant without a "rounding scheme" for the fractional shots, in which case one is driven back to a heuristic approach.

Another approximation approach has promise, however, and will be called the ALIAS algorithm. The concept behind ALIAS is to approximate (2) and (3) by (4) only for the purpose of getting an allocation of shots to TOs, and to search for a maximum of (2) and (3) using the allocations computed in (4).

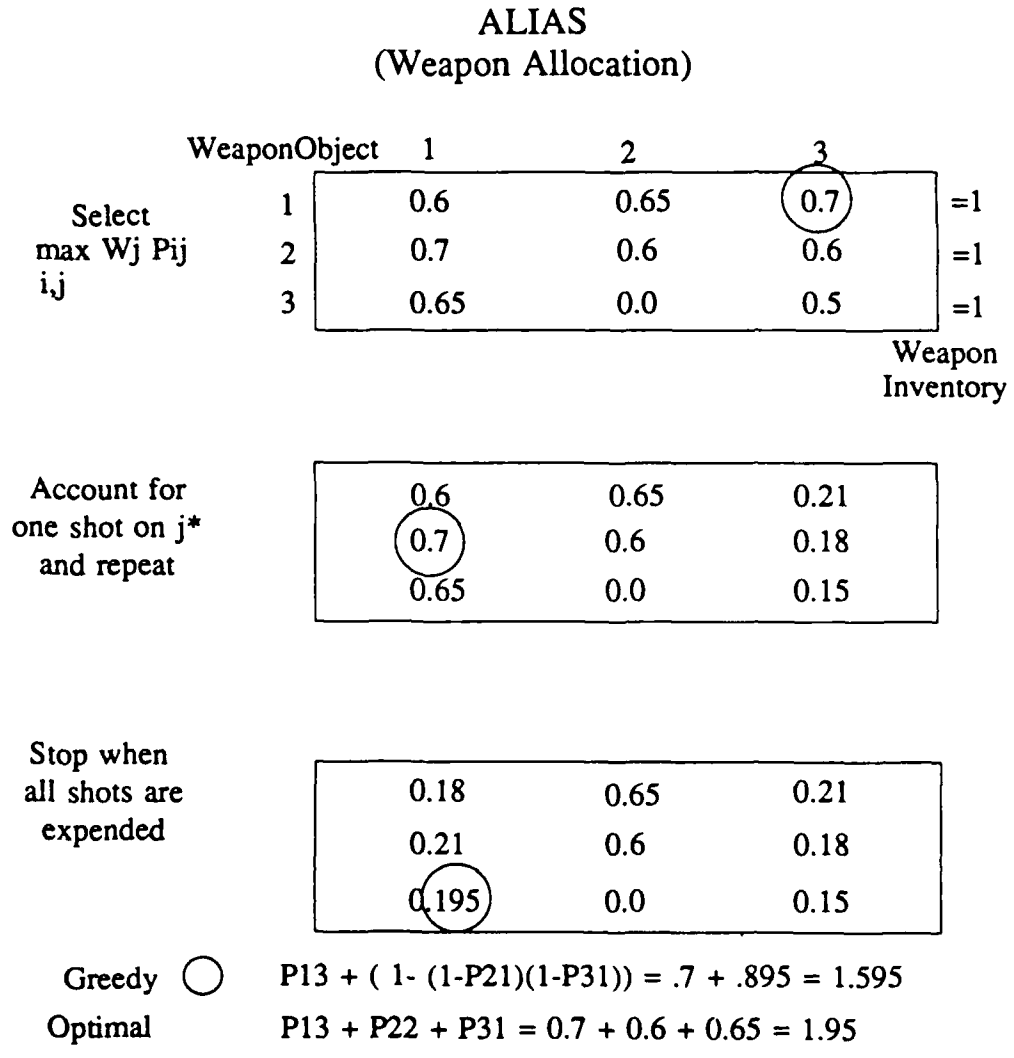


Figure 6. Example of solution of ALIAS weapon allocation heuristic.

The implementation of ALIAS using the Greedy algorithm on (4) considers each TO in turn potentially yielding the maximum value to the objective function without regard to shooting at all TOs aimed at an asset. Some TOs may receive multiple shots while other TOs for the same asset are not shot even once. Each TO is allocated some number of shots based upon the kill probability and TO weight. An allocation of shots on TOs from (4) does not give the objective value in (2) or (3) of multiply targeted assets, and so the true value of the allocation must be computed.

i. Compute the weight of each TO, w_j

$$w_j = w_k \; / \sum_{j \in J_k} n_j \; for \; all \; j$$

and set $\alpha_{ijt} = 0$ for all i, j, and t. The ALIAS objective function (4) uses the weights of the TOs,

not the weights of the assets. Each TO is assumed to have equal impact on the survival of its asset, and thus a TO carries the average weight of TOs on an asset.

ii. Find the maximal element (indexed by *i, j, and t*) given in (7) of all feasible elements (the unconstrained shots, in 7a) for entry into the Greedy solution

$$s\,(\,i^*,\ j^*,\ t^*\,) = \max_{i,j,t} \beta_j\ \xi_j\ w_j\ p_{ijt} \tag{7}$$

subject to

$$\sum_{t=0}^{T} \sum_{j \in J_k} \alpha_{ijt} < I_i \ :\ \textit{inventory constraint} \tag{7a}$$

$$\sum_{j \in J_k} \sum_{i \in I} \alpha_{ijt} < b \ :\ \textit{tactical planning budget constraint}$$

$$\sum_{j \in J_k} \alpha_{ijt} < c \ :\ \textit{refire constraint}$$

Eventually each TO is allocated some number of shots via this step based upon the kill probability and threat weight.

iii. Having now made a shot at TO j^*, revise all kill probabilities for TO j^* by this shot's miss probability for j^*. Compute $p_{ij^*t} \leftarrow (1-p_{i^*j^*t^*}) \times p_{ij^*t}$ for all i, t.

iv. Increment by one the shot allocation of weapon i^* on TO j^* in time t^*. Set $\alpha_{i^*j^*t^*} \leftarrow \alpha_{i^*j^*t^*} + 1$.

v. Repeat ii - iv until step ii fails (a maximum of M iterations).

vi. Calculate (2) or (3) using the ALIAS allocation α^* of (4) as the real allocation for (2) or (3) giving $S(\alpha^*)$ and return to asset selection algorithm.

The implementation of ALIAS using the Greedy algorithm considers each TO in turn yielding the maximum value to the "approximate" objective function (4) without regard to shooting at all TOs on an asset as required by (2) and (3). Some threats may receive multiple shots while other TOs for the same asset are not shot even once. This has a major impact on the asset selection algorithm as some assets will have a cover, some assets might be uncovered with shots, and still other assets might have no shots at all. The asset selection algorithm must recognize this situation in its determination of the next set of assets to try to defend.

Figure 6 presents a very simplistic numerical example of the ALIAS algorithm where 3 shots are budgeted at 3 TOs. The first rectangle (Pk matrix) gives the weapon-object expected value, i.e., the expected value of disabling the object (column) by a shot from the weapon platform (row). Each weapon only has one shot, each object is destined to a unique target, and each target has a value of unity; thus the expected value in the table is just the kill probability itself. The first "*ij*" selected lies in row 1 column 3, i.e., weapon 1 fires its only shot at TO 3 with a kill probability of 0.7. The kill probabilities in column 3 are corrected to reflect the reduced probability of TO 3 getting through the defense.

In Pk matrix #2, we observe that the next "*ij*" chosen lies in row 2 and column 1 with a Pk of 0.7. Note that even though there exists two "0.7"s row 1, column 3 was selected first, and therefore is the first to enter the solution. The kill probabilities in column 1 are corrected to reflect the reduced probability of TO 1 getting through the defense.

In Pk matrix #3, we observe that the next "*ij*" chosen lies in row 3 and column 1 with a Pk of 0.195, even though higher Pks exist in column 2. Column 2 can not be chosen because the two weapons applicable to column 2 have exhausted their inventory. The ALIAS stops as the 3 budgeted shots have been used with a solution computed to be 1.595, a solution less than the optimal 1.95 (by observation). This is a not uncommon behavior for a Greedy type algorithm, i.e., to miss an opportunity at a significantly better solution due to its looking only one move at a time, but the ALIAS is *fast!* The following algorithm corrects to some degree the shortsightedness of the Greedy algorithm.

4.2.3 Greedy Backtracking -- The PAIRswap Algorithm In general a backtracking algorithm revises the incumbent (incomplete) solution by reviewing (backtracking) contingent changes to the incumbent solution to permit the inclusion of the current candidate variable such that the change has the greatest (least) marginal return. Forward backtracking starts this process by considering the first variable to enter the solution, then the second, etc. Backward backtracking starts this process by considering the last variable to enter, then the next to last, etc. A level one backtrack will make the one best change and consider no additional changes, while a full backtracker will make all changes possible (with a test for cycling between changes, which may occur over several iterations). The drawback to a pure backtracking algorithm, besides the run time, is that all intermediate solutions are incomplete.

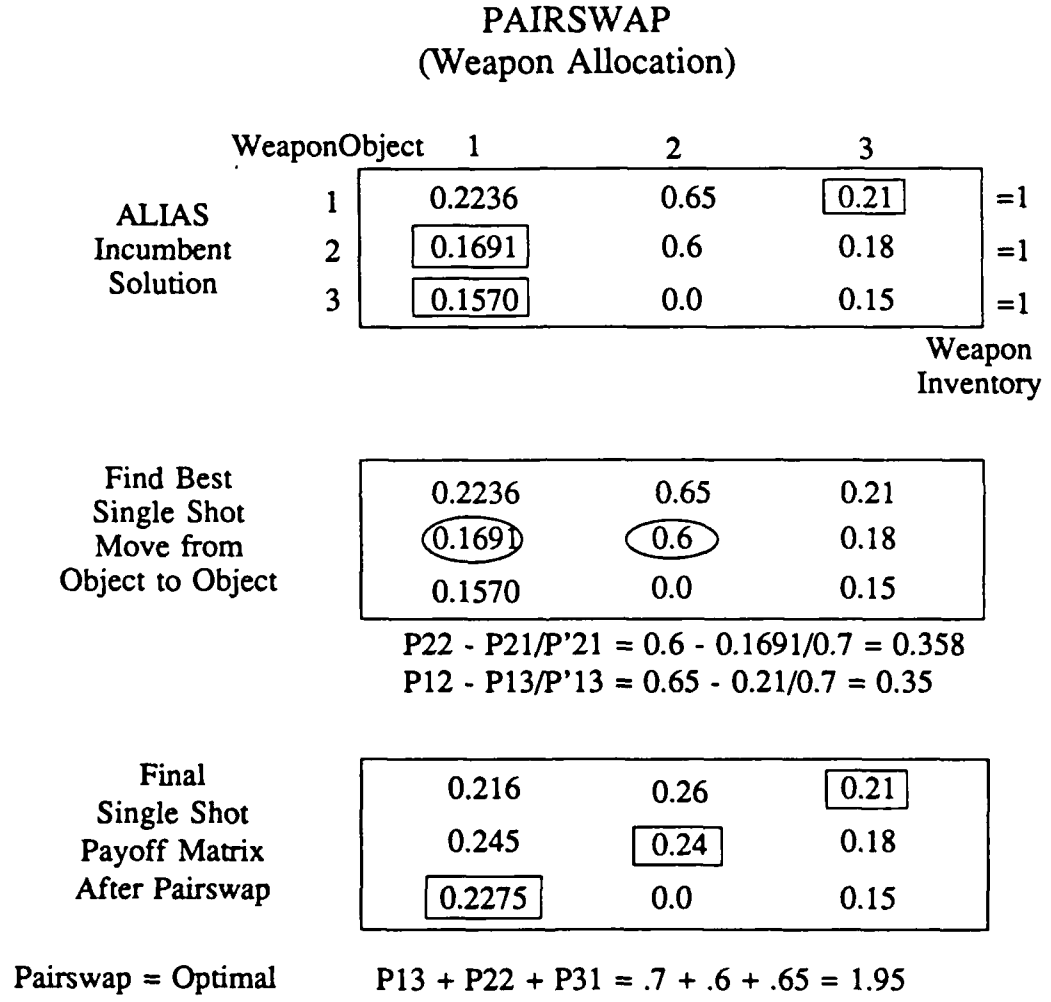


Figure 7. Example of solution of PAIRswap allocation heuristic.

Our implementation of a "backtracking" algorithm is only one level and begins only *after* a complete solution exists, therefore, it is far enough from a pure backtracker that we refer to it by a different name. We only consider a one level change, i.e., a pairwise swap of shots from one weapon-object allocation to another weapon-object allocation such that the marginal value for the exchange is (most) positive. The PAIRswap weapon allocation algorithm takes the ALIAS solution as its starting point and iteratively looks for the one best shot swap that will maximize the objective function (4). The objective functions of (2) and (3) have been tried but the extra computation time was not worth the higher quality answer. For example, a one percent increase (or decrease) in the expected surviving value is not worth twice the computation time.

Two cases are considered by the PAIRswap algorithm. Case 1 occurs when it is possible to *move* one shot from a weapon-object to another weapon-object. This is the instance presented in Figure 7. Note that object 2 did not get a shot because the Greedy algorithm blindly picked the highest Pk shot weapon 2 on object 1 in iteration #2, and then in iteration #3 had to pick a second shot on object 1. The first matrix in Figure 7 is the result of the ALIAS algorithm on the example given in Figure 6. The PAIRswap algorithm first chooses the incumbent shot at P13 and tries to *move* it to P12 yielding a marginal return of 0.35; or moving it to P11 (not evaluated for obvious reasons). The next incumbent shot to be considered is that of P21 being moved to P22 yielding a marginal return of 0.358. The final shot to be considered is P31 to P22 yielding a value of 0.0 or to P31 (not evaluated). The one shot best *move* is the shot at P22 (chosen *before P31* by ALIAS), which yields the optimal answer. PAIRswap stops when no moves can be made.

Case 2 involves *swaps* instead of *moves*. PAIRswap searches for the one shot best swap at each iteration. A *swap* is a *move* that may involve changing the weapon and period, and because of the constraints, requires another *move* to free a "space" for the shot. So two (a pair) of moves (swaps) must occur, hence the name pair swap (swap a shot from a pair of weapon-object-period allocations). All possible quadruples are searched at each iteration two find the one best pair of swaps. Of course, in the implementation, a lot of filtering of possible quadruples can occur.

Like the Genetic Search algorithm, the PAIRswap algorithm is massively parallel, and run times can be reduced by $1/n$ when there are n processors.

4.2.4 Greedy MARGinal Returns -- The MARG Algorithm Another implementation of the Greedy to solve (2) and (3) is to use the marginal effects on the objective function as the selection criterion for adding a shot at each iteration. When adding a shot does not violate a shot limitation constraint the difference in the value of the objective function before and after the shot is the marginal effect, but if the addition of a shot does violate a constraint the marginal effect is determined by finding the currently allocated shot to "swap out" for the one under consideration which will yield the best marginal change in the objective function.

The allocation process can start with no shots allocated, and continue iteratively adding the most marginally effective until the increased effectiveness of the next best shot is below some pre-set stopping point. Alternatively, the process could start with an initial non-zero allocation, and proceed from there (the initial allocation could be that from the previous cycle).

There are a few complications with this approach. The objective (2) is not a "smooth" curve of discrete points, unfortunately. If, for example, ten objects are targeted on asset k, we get no survival value until all ten have at least one shot directed at them. When all objects going to an asset have one shot allocated to them then the survival value from that asset jumps discontinuously from 0 to $w_k \prod_{j \in J_k} p_j$ (where p_j is the kill probability of the one shot at object j). We avoid this problem by artificially considering each targeted asset to be divided into as many subsites as there are objects going to it, with each subasset having one object associated with it, and carrying $1/N_k^{th}$ the asset weight (where N_k is the number of objects going to asset k). As the algorithm proceeds, and shots are allocated to objects, if ever an asset's objects are all allocated shots the asset's artificial subdivision is abandoned, as the marginal survivability of added shots on objects going to that asset becomes meaningful at that point. One can think of the problem as that of climbing a hill whose approaches are blocked by a sheer cliff. The subdivision technique works by laying an inclined plane up to the top of the cliff, and using it to move up to the hill to be climbed. Once the hill is reached the plane can be removed.

Another complication is the fact that when a shot is allocated to an object that is targeted at an asset, the shot affects not only the marginal survivability for all subsequent shots on that same object (because the survival probability of the object is diminished), but also the marginal changes in the survivability due to shots aimed at all other objects targeted at the same asset (because the overall probability of the asset's survival is affected also). (Because the additional allocation of a shot changes the "nearby" marginals, we can describe the effort to find the maximum allocation as "hillclimbing on a rubber sheet.") Since the addition of more than one shot may change the associated marginals so much that a new greatest marginal should be determined before going too far in the current direction, acceleration

techniques that add several shots per iteration cannot be employed; each iteration only adds a single shot to the allocation. If we have a set of allocations α_{ij}, the marginal change in the objective function of (3) if α_{xy} is increased by 1 (where $y \epsilon J_k$) is:

$$w_k \prod_{\substack{j \epsilon J_k \\ j \neq y}} \left[1 - \beta_j \xi_j \prod_{i,t} (1 - p_{ij}(t))^{\alpha_{ij}(t)} \right] \prod_{i,t} (1 - p_{iy}(t))^{\alpha_{iy}(t)} p_{xy}(t).$$

There are some variations of the marginal return greedy that may prove effective. One is to keep track not only of the shot with the highest marginal return at each iteration, but also the second best shot. A random variate, drawn according to probabilities that depend on the relative difference in these two marginal values, could determine whether the best or the second best gets picked for inclusion at an iteration. (If the two values were the same, the probability of either shot being picked is one half; the chance of the second best shot being picked would decrease as the relative difference gets bigger.) The final allocation would depend, in part, on chance. Many of these chance-influenced allocations can be determined concurrently and the best picked.

Another variation is a generalization of the greedy algorithm. If instead of considering shots (a combination of a single row and column -- weapon and TO) at an iteration, consider sets of k rows and K≥k columns. Using marginal returns as the assignment values, the assignment of a single shot from each of k rows so that each of the K columns has no more that a single assignment is the easily and quickly solved assignment problem. The best of all subsets of k rows and K columns in an iteration is a subassignment that can be allocated just as a single shot is in the previously described greedy algorithm (indeed, the original greedy is a specific case of this where k=K=1). The other extreme is where k=all rows and K=all columns, in which at each iteration we find the best suballocation where each TO gets no more than one shot. It would be useful to see how the quality of the solutions is affected by changing the parameter k.

Because of the target subdivision artificiality introduced to allow initial allocations to multiply-attacked targets, at the end of the greedy algorithm there may be one or more assets that only have partial coverage (possibly because the inventory of shots has been used up). A small postprocessing function strips away the shots from each of these uncovered assets in turn (shots at a partially covered asset would be wasted anyway) and allocates these in the same fashion as before to the other assets. This postprocessing continues until all assets are either totally covered or totally undefended by shots (no partial coverage).

The greedy and its modifications can all be imbedded in a larger allocation process that first decides which of the targets to defend and which to abandon.

5. SOLUTION MEASURES OF EFFECTIVENESS

Because the global optimum in (2), (3), and (4) cannot be known with certainty, we are faced with determining some metrics by which to compare alternative algorithms - solution quality and computer performance. This section explores some useful measures of effectiveness.

By categorizing the model based on a sequence of simplifying assumptions we can consider the possible solution approaches for each category. Obviously the more tractable the model, the better chance of a useful solution. But all solution approaches should be judged by certain standards that will measure their usefulness or effectiveness. We need to decide what constitutes "effectiveness," determine how to measure it, and derive some bounds on these measures. We have taken a preliminary look at how to specify algorithm performance and have found that the measures tend to fall into two categories: measures of solution quality, and measures of the computational resources need to achieve a solution.

The obvious measure of the usefulness of an algorithm is the quality of its solution -- how does it compare with the optimum. Here there is a measurement difficulty because to find the optimum entails solving the NP-complete problem which is totally impractical for large problems (and even problems many would consider small, e.g., a 15x15 multiple assignment problem). The approach we will use is to compare solutions obtained using candidate heuristic algorithms for large problems, as well as comparing them with the optimum for problems which are small enough to solve optimally. In some

cases a "worst-case" analysis will be performed to determine how far from a possible optimum an algorithm might be.

The robustness of an algorithm should also be considered as a measure of its usefulness. This measure, though, is very difficult to quantify. What we will do is run the candidate algorithms through a wide range of parametric variations, and note the effects of these variations on the other measures. A robust algorithm is one that is useful (in terms of the other measures) over a wide range of problems. We can at least eliminate candidate algorithms that are dominated by others in the sense that they are no better than other candidates for any situation, and are worse in some situations.

One measure of usefulness is algorithm run time. An algorithm ought not run longer than the cycle time between weapons firings and the cycle time between data updates (because in the former case we would be wasting shot opportunities, and in the latter case we would be wasting information).

Another measure of usefulness is computer memory requirements. An alternative that can serve to reduce memory requirements is to increase the I/O requirements to mass storage devices, which increases run times; there is always a tradeoff between speed and program size. There is also a limit to the data transfer rates into and out of a direct access storage device.

If strict limits to available memory or run time are not available, it would also be useful to quantify algorithms by their efficiency: the quality of the solution over the resources (time or size) used.

We intend to determine the appropriate algorithms for each category of objective function, recognizing the effects of simplifying assumptions in each case. Each candidate will be tested parametrically, first in an effort to "hone" and improve it, and then to compare it with others. The parameters that will be varied for each candidate include the number of shots, the number of TOs, the weapon refire times, and the functions relating kill probabilities with range. A worst-case analysis will be performed where appropriate. In the end we will have a set of reasonable, feasible, and useful algorithms, each identified with a range of parametric values where they are viable. We will also have ranges of parametric values within which reasonable and effective shot assignment algorithms exist, and ranges where none of our algorithms work. This information may be useful for system designers and architects who may have control over the parameter ranges that will be in effect.

6. SUMMARY

The role of weapons allocation for a battle manager in a fairly diverse battle engagement environment *after* missile launch has been modeled by a set of three successively simpler formulations. Sets of assumptions that underly each of our specific weapons assignment models have been listed, along with the restrictions on the algorithms that could be used in each case. The algorithms that will be used must meet certain performance requirements, which are identified. Our algorithms can solve models (2) through (5) and operate in an iterative fashion between asset selection and shot allocation given a set of assets to defend. The algorithms have been designed to allow most combinations of asset selection algorithm and allocation algorithm. Finally, we tell how we will analyze the algorithms to compare them, one against the other in the differing situations.

REFERENCES

1. Read, W.T. *Tactics and Deployment for Anti-Missile Defense*, Bell Telephone Laboratories, Whippany, N.J.

2. Burr, S.A., Falk, J.E., and Karr, A. F., *Integer Prim-Read Solutions to a Class of Target Defense Problems*, Journal of Operations Research, V33/4, August, 1985.

3. Gabow and Tarjan, AT&T Bell Laboratories, Murray Hill, N.J., 1986.

4. Lloyd, S. P. and Witsenhausen, H. S., *Weapons Allocation is NP-Complete*, IEEE Summer Simulation Conference, 1986.

5. Lloyd, S. P. Personal communication, AT&T Bell Laboratories, Whippany, N.J., March 1987.

6. Maffioli, F., "Randomized Algorithms in Combinatorial Optimization: A Survey", in *Discrete Applied Mathematics*, North-Holland, 1986, pp 157-170.

7. McRoberts, K.L., "A Search Model for Evaluating Combinatorially Explosive Problems", *Oper. Res.*, 1971, 19:1331-1349.

8. Holland, J. H. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

9. DeJong, K.A., "Adaptive System Design: A Genetic Approach", *IEEE Transactions on Systems, Man, and Cybernetics SME-10*, 1980, pp. 566-574.

10. Bethke, A.D., *Genetic Algorithms as Function Optimizers*, Ph.D. Thesis, University of Michigan, 1980.

11. Holland, J.H., Holyoak, K.J., Nisbett, R.E., and Thagard P.R., *Induction: Processes of Inference, Learning, and Discovery*, MIT Press, Cambridge, Mass, 1986.

12. Hausmann, D., Korte, B., and Jenkyns, T. A., "Worst case analysis of Greedy Type Algorithms for Independence Systems", *Mathematical Programming Study*, 12, North-Holland, New York, 1980, 120-131.

13. Garey, M.R. and Johnson, D.S., *Computer and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.